Computers Math. Applic. Vol. 23, No. 6–9, pp. 477–491, 1992
Printed in Great Britain.

0097-4943/92 $5.00 + 0.00

COMPLETING SORT HIERARCHIES

ANTHONY G. COHN
Division of Artificial Intelligence, School of Computer Studies
University of Leeds, Leeds LS2 9JT, England

Abstract—This paper discusses the structure of sort (or *is-a*) hierarchies. The effect of different kinds of such hierarchies on different kinds of many sorted logic is discussed both from the point of view of expressiveness and computational efficiency in resolution theorem proving. A technique for improving the computational efficiency of inference when the sort hierarchy is incompletely specified is suggested, which involves embedding the hierarchy into a complete Boolean lattice without losing information on under specified sorts. Such an embedding can be used for any poset or lattice structured hierarchy.

1. INTRODUCTION

A sort hierarchy in a many sorted logic is akin to the *is-a* abstraction hierarchies used in most semantic network systems,[1] such as the type lattices of conceptual graphs [1] or the various kinds of TBOX (terminological box) used in the KL-ONE family of hybrid systems, e.g., KRYPTON [2]. A many sorted logic is one in which the universe of discourse is divided into subsets (called *sorts*) rather than being one homogenous set. This is achieved by specifying a set S of *sort symbols*[2] each of which denotes a nonempty subset of the universe. Function and predicate symbols may be declared as only applying to particular argument sorts, and function symbols may in addition be declared as returning individuals of a particular sort. The sort structure and these declarations are usually called the *signature*. *Sortal constraints* may be attached to variables—in this paper we will always indicate that a variable x is of sort τ by the notation $x:\tau$. The effect of this is simply to restrict the quantification of x to τ. For example, $\forall x:\tau P(x)$ makes a claim only about all individuals in x's sort τ, not about all individuals in general. Formulae have to be such that the sorts of terms *match*[3] the sorts of the argument positions they occur in—formulae must be *wellsorted*. The usual motivation quoted in the computational logic literature for using a many sorted logic is that of efficiency: the search space of a problem in automatic theorem proving can be dramatically reduced [3–7]. Other advantages of using a many sorted logic include the mental hygiene of declaring the sortal behaviour of the (non logical) symbols (cf. the advantages claimed by software engineers of using typed programming languages) and the utility of wellsortedness for a simple but efficient integrity check on knowledge base updates and queries. For a general introduction to the many sorted logic literature see Cohn [8].

Originally [9,10], sorts were disjoint (i.e., their interpretations were pairwise non intersecting sets). However, in most computational many sorted logics now used for AI, sorts may overlap and include one another; this is usually achieved by specifically declaring a transitive *ordering* relationship, $\sqsubseteq$, between pairs of sorts: which is reflected in the model theoretic semantics by making the denotation of one sort a subset of the other. Such many sorted logics are sometimes known as *order sorted* logics for this reason. This paper is devoted to a discussion of the nature

I am grateful to all those with whom I have ever discussed many sorted logic. In this instance particular thanks are due to Uli Hedstueck and Gert Smolka. I also wish to thank Fritz Lehmann and the anonymous referees who commented on an earlier draft of this paper. The financial assistance of the SERC under grants GR/C/65148 and GR/F/64380 is gratefully acknowledged. Part of this research was conducted during a visit to IBM (Stuttgart).

[1] It should be pointed out that in many sorted logics and in this paper we are only concerned with definitional hierarchies, i.e., those without defaults.

[2] We will often be informal and talk of 'sorts' when strictly we should have said 'sort symbols.'

[3] We will discuss below precisely what 'match' might mean.

of the knowledge represented in sort hierarchies and its effect on the inference machinery of resolution-based automatic theorem proving, a common method of automated reasoning in AI.

The structure of the rest of the paper is as follows. First we will discuss the kinds of information that may be represented in sort (or equivalently, *isa*) hierarchies. Then we will discuss the nature and effect of different sort hierarchies on so called ***non substitutional*** many sorted logics. The main result of the paper is to show how an incompletely specified sort hierarchy for a non substitutional logic can be embedded in a completely specified hierarchy (that is, in a complete Boolean lattice) and still retain the same set of valid theorems, but with greater computational efficiency.

2. KINDS OF HIERARCHY

A sort hierarchy is a partially ordered set (poset) of sorts. This poset may or may not be a true mathematical lattice or a true Boolean lattice. Often, the only information given explicitly about the sort hierarchy is just the ordering $\sqsubseteq$, mentioned above, which defines a partial ordering on S. However, we want to define a greatest lower bound (glb) operator,[4] $\sqcap$ on such hierarchies to be used as the matching or unifying operator for sorts. The interpretation of $\sqcap$ is not necessarily $\cap$—the denotation of the glb of two sorts may be a *strict subset* of the intersection of what the two sorts denote. Figure 1 illustrates the situation. Although τ_3 is the greatest lower bound of τ_1 and τ_2 in the hierarchy, because all we know is that τ_3 is a subsort (and thus, semantically, a subset) of both τ_1 and τ_2, the model illustrated in the Venn diagram is perfectly feasible.

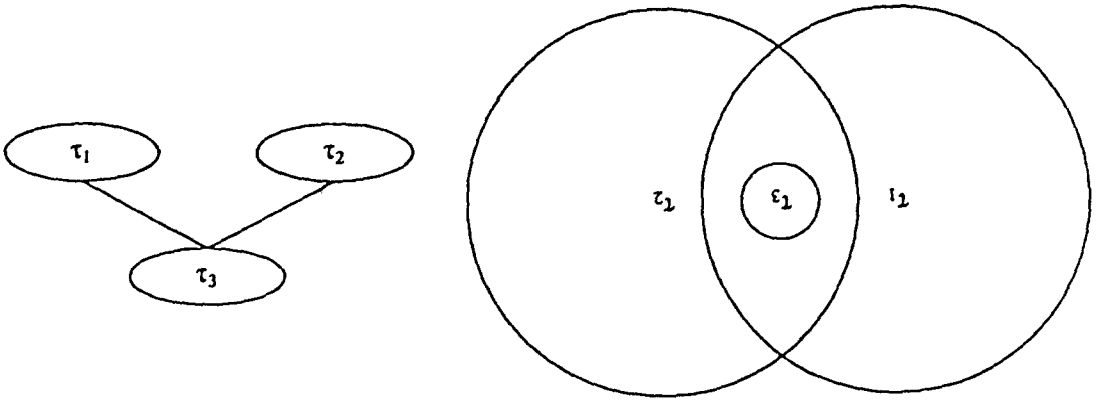

Figure 1. τ_3 is the glb of τ_1 and τ_2, but it may describe a smaller set than the intersection of τ_1 and τ_2.

However, sometimes a glb operator is introduced directly into the description language for S and is interpreted strictly as set intersection. In such many sorted logics one can write down not only that $\tau_1 \sqsubseteq \tau_2$, but also that $\tau_3 = \tau_1 \sqcap \tau_2$. If $\tau_3 = \bot$ (i.e., the empty, 'bottom' sort), then this is equivalent to saying that τ_1 and τ_2 are mutually disjoint. If the glb of every pair of sorts in S is known and distinct (unless equal to $\bot$) and every sort τ has a complement (i.e., another sort meaning 'not τ'), then the sort hierarchy has the structure of a complete Boolean lattice, with 2^n nodes where n is the number of nodes just above $\bot$.

Knowing the structure of S completely as a Boolean lattice has computational advantages: for example one can use efficient bit encoding techniques [11]; in the simplest case a string of n bits represents one of the 2^n nodes. Furthermore, inference rules can be made more powerful by taking advantage of the completeness of the sort hierarchy; reasoning by cases [12] is possible: for example, one can reason that a term is not of a particular sort by considering each subsort. However, in some domains one may not have complete information available at the time of declaring the sort hierarchy; this may occur particularly in Natural Language understanding applications [5,13]. However if *sort literals*[5] (i.e., literals whose predicate name is a sort symbol)

[4] For simplicity, we will ignore any complications caused by the greatest lower bound not necessarily being unique in a poset. In fact, the embedding specified below handles this situation without amendment.

[5] Sort literals are sometimes called *characteristic literals* since a sort predicate is the characteristic predicate for the set denoted by the sort. For further discussion of characteristic literals in many sorted logics see [12].

or equality literals (i.e., literals which predicate the equality of two terms) are present in the axiomatisation [12], then these can affect the semantic structure of the sort hierarchy as we shall see in the next section. In this case the rules of inference have to be rather weaker than in the case where the sort hierarchy is complete or than when characteristic literals are forbidden.

A general way of specifying a sort hierarchy is axiomatically. For simplicity we will use *clausal* form, i.e., formulae are disjunctions of literals and all variables are universally quantified. Let Δ be the set of sort symbols. Then axioms of the form $\forall x \neg\tau_1(x) \vee \cdots \vee \neg\tau_n(x) \vee \tau'_1(x) \vee \cdots \vee \tau'_m(x)$ where $\tau_i, \tau'_i \in \Delta$, give information about the sort hierarchy. It is worth enumerating some of the special cases:

1) $n = 1$, $m = 1$: $\tau_1 \sqsubseteq \tau'_1$.
2) $n > 1$, $m = 1$: $\sqcap\{\tau_1, \ldots, \tau_n\} \sqsubseteq \tau'_1$.
3) $n = 1$, $m = 0$: impossible, τ_1 can't be a (nonempty) sort.
4) $n = 0$, $m = 1$: τ'_1 is the universal sort, $\top$.
5) $n = 0$, $m > 1$: $\tau'_1 \ldots \tau'_m$ cover the universe.
6) $n = 1$, $m > 1$: $\tau'_1 \ldots \tau'_m$ cover τ_1.
7) $n = 2$, $m = 0$: τ_1 and τ_2 are disjoint.
8) $n > 2$, $m = 0$: τ_1 is the complement of $\sqcap\{\tau_2, \ldots, \tau_n\}$.
9) $n > 1$, $m > 1$: $\tau'_1, \ldots, \tau'_m$ cover $\sqcap\{\tau_1, \ldots, \tau_n\}$.

Notice that cases (1), (2), (4) are *definite* clauses. This will be of interest below.

3. MANY SORTED LOGICS

We will now consider the use of sorts in automatic theorem proving and resolution based inference systems in particular. We can regard a many sorted logic as consisting of two components, the signature and the axiomatisation. We can view the signature as consisting of axioms of a special kind, where all the predicates are sort symbols. This component is analogous to the TBOX (or Terminological Box) of KL-ONE like systems such as KRYPTON. The other component is the ABOX (or Assertional Box). Since we are dealing with many sorted logics here, we will call the first component an SBOX (or Sort Box). The SBOX just contains information about the sort hierarchy (and which terms are of which sorts) while the ABOX can contain arbitrary information.

In general, the sort hierarchy can interact with the main axiomatisation in such a way that the hierarchy is effectively (semantically) altered.[6] The main way this can happen is through the use of explicit characteristic literals in the main axiomatisation.[7]

An example of this kind of semantic restructuring of the SBOX by the ABOX is given by Beierle *et al.* [15]. Consider the sort structure shown in Figure 2. Suppose $\forall x{:}\, S4\ S3(x)$ is asserted or provable from the ABOX, then we know semantically (rather than syntactic point of view of the SBOX declarations), that the SBOX hierarchy is effectively changed as depicted in Figure 3. As Beierle *et al.* point out, it is very easy to lose completeness when taxonomic information can be specified both in the sort signature (the SBOX) and in the main body of the logical axiomatisation (the ABOX). When completeness is retained, they term the logic *closely coupled.* For example they show that the language KRYPTON [16] in the KL-ONE family of semantic

[6] Frisch [14] has investigated a class of logics, (which he terms *substitutional* many sorted logics) where this cannot happen because sorts are only allowed in the ABOX as restrictions on variables. An interesting result concerning substitutional logics is that given a sound and complete unsorted logic, Frisch shows that it is trivial to turn it into a sound and complete many sorted calculus within the substitutional framework. For non substitutional logics proving soundness and completeness may be non trivial. However this only works providing the SBOX has a *unique minimal model.* The only obvious syntactic criterion to ensure this is the case (a sufficient though not necessary condition) is that the SBOX be represented using definite clauses only—i.e., we could only use clauses of the form (1), (2) and (4) above. So, in the substitutional framework, we are then (in practice) also committed to not being able to specify arbitrary knowledge in the sort hierarchy, and moreover cannot express it in the ordinary axiomatisation either (the ABOX) because of the ban on sort predicates there. Thus, essentially the only kind of information specifiable in a substitutional many sorted logic is subsort information. Because of this inflexibility and because we cannot express sort information in the ABOX (by definition), we will not consider such many sorted logics further here.

[7] It may also happen by use of equality literals as noted by Walther [17]. By writing $\exists x{:}\tau\ x = \alpha$ one can express that α is of sort τ which is just what a sort literal $\tau(\alpha)$ expresses.

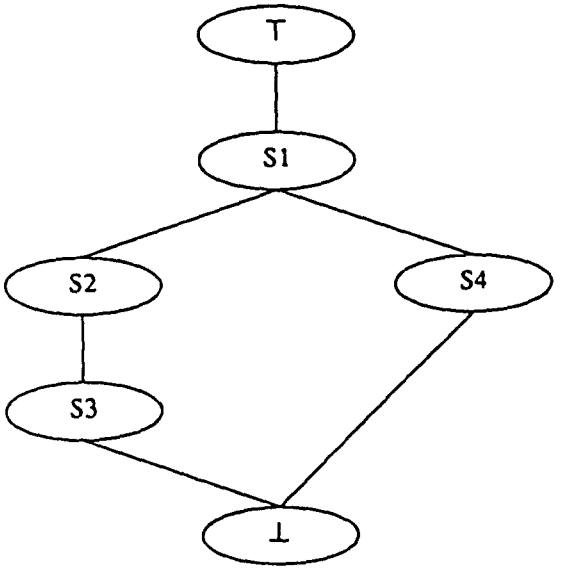

Figure 2. An arbitrary sort hierarchy specified in the SBOX (Sort Box). ⊤ is the 'top' sort which represents the universe of discourse and ⊥ is the empty 'bottom' sort.

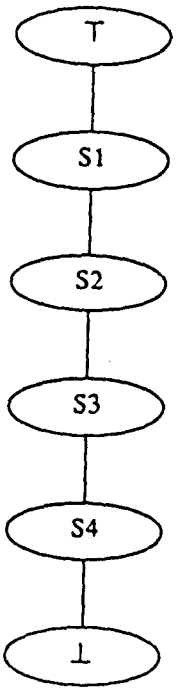

Figure 3. The altered hierarchy of Figure 2 which results from asserting $\forall x$: $S4$ $S3(x)$ in the ABOX, which effectively means the same as specifying $S4 \sqsubseteq S3$ in the SBOX.

network languages is not closely coupled. We will only be concerned with closely coupled logics in this paper.

It is possible to design a many sorted logic which allows characteristic literals and where the sort hierarchy can be incompletely specified but there would be appear to be computational problems in so doing. The only existing logic with these properties is that of Beierle *et al.* [15]. We will discuss the problems of this logic in detail below and contrast it with the author's own logic, LLAMA [4].

Beierle *et al.*'s many sorted logic has a sort structure which consists just of subsort declarations but characteristic literals are allowed in the main axiomatisation. The logic is obtained by changing standard resolution based, many sorted logic with a subsort ordered hierarchy (such as that of Walther [17]) in three ways. First, the unification algorithm has to be changed, secondly, a rule for resolving characteristic literals which differ in the predicate name is introduced, and thirdly, a rule for eliminating negative characteristic literals whose argument is a subsort of the predicate is necessary.

We will consider the changed unification algorithm first. Unification is an operation which finds a substitution (i.e., a replacement of terms for variables) which when applied to a set of expressions E, makes E a singleton. We will represent a substitution as a set of substitution components each of the form t/α, where t is a term and α the variable being substituted for. Unification is used during the resolution rule of inference in order to make literals from different clauses complementary (e.g., the substitution $\{a/x\}$ makes the first literals in the two clauses $P(x,a) \vee Q(x,x)$ and $\neg P(a,a) \vee R(b)$ complementary, enabling a resolvent of $Q(a,a) \vee R(b)$ to be inferred). As is usual, we will only consider *most general unifiers*. In Walther's unification algorithm, many sorted unification simply returns a well sorted substitution (i.e., a replacement of terms for variables where the sorts of all the terms are a subsort of the sorts of the variables being substituted for). However, in Beierle *et al.*'s logic, when unifying two terms $t1$ and $t2$, the many sorted unification algorithm returns a pair $\langle \sigma, SL \rangle$ where σ is the standard (well sorted) unifier of $t1$ and $t2$ and SL is a conjunction of positive sort literals. These literals give any conditions on the sorts of terms (which cannot be expressed as direct constraints on variables) in order to ensure the unification is correct. These will be introduced when unifying variables of

differing sorts, or sometimes when unifying a non variable term[8] and a variable of differing sorts, as discussed below. Rather than give the details of the algorithm here, an example will provide sufficient information to understand the difficulties raised by the method. Given two variable terms $x_1 : \tau_1$ and $x_2 : \tau_2$, the returned unifier is $\langle \{z\colon \top/x_1,\ z\colon \top/x_2\},\ \tau_1(z\colon \top) \wedge \tau_2(z\colon \top)\rangle$. The important point to note is that the new variable introduced is of sort $\top$, meaning the universal sort including everything. The correctness argument lying behind this perhaps counterintuitive and certainly unusual many sorted unification algorithm is that we cannot simply attach a sort constraint of τ_3 to z where τ_3 is the normal greatest lower bound of τ_1 and τ_2 in the sort hierarchy, because the denotation of the sort τ_3 may be interpreted in the model domain as a *strict subset* of the intersection of the denotations of τ_1 and τ_2 instead of being interpreted as the intersection itself. Figure 1 illustrates the situation. Although the true intersection sort is 'invisible' in the sort hierarchy it is what is needed in order to unify the two given terms in the most general way. Thus attaching a sort constraint of τ_3 to z would be overly restrictive.

The solution adopted is not to restrict z at all by conventional sorting techniques (thus it has sort $\top$) but rather to include the negation of the SL condition in the second part of the unifier, $\neg\tau_1(z\colon \top) \vee \neg\tau_2(z\colon \top)$ in any clause to which the substitution is applied. For example, suppose we were resolving $P(x : \tau_1) \vee Q(x : \tau_1)$ and $\neg P(y : \tau_2)$, then the resolvent would be $Q(z : \top) \vee \neg\tau_1(z : \top) \vee \neg\tau_2(z : \top)$. The negative sort literals predicating z ensure that z must be both a τ_1 and a τ_2. Although this solves the problem of potentially losing completeness had we simply attached the sort τ_3 to z, the technique looks disastrous from a computational viewpoint because we lose all the control on the search space derived from sortal constraints on variables, which is the classic computational advantage of many sorted logic, as is well documented [3,7,17–19]. Giving a variable the sort $\top$ means that every term can be substituted for it. As an attempted proof progresses and variables are unified against other variables, more and more variables in resolvents will be labelled with sort $\top$. Moreover, formulae will contain negative characteristic literals predicating variables. Although one would hope that most terms would be declared to be of the appropriate sort via sort declarations, if there are ways of proving that terms are of particular sorts, (i.e., there are positive occurrences of characteristic literals in the axiomatisation) then such negative characteristic literals may act as generators and cause an explosion in the search space. The situation is particularly bad when the variable predicated occurs elsewhere in the wff, for if the characteristic literal is resolved away first, instantiating the variable in so doing, and an inappropriate choice is made, then the literal(s) containing the other occurrences of the variable may be unprovable causing backtracking in a depth first search or needlessly exploring a branch of a breadth first search space. It may be possible to retrieve the situation and still obtain much of the reduced search space of a many sorted logic through the use of control rules [20] but then we might as well admit that we are effectively abandoning many sorted logic.

It would appear that a slightly better solution (not mentioned by Beierle *et al.*) would be to make z of sort τ_1 (or τ_2) and then simply predicate τ_2 (or τ_1) of z in the SL condition. However, this is again not computationally attractive because negative characteristic literals predicating a variable are still present in formulae.

It is worth pointing out the distinction between the literals introduced through the SL condition in this logic and the *prosthetic literals* of the many sorted logic LLAMA [4]. In the case of LLAMA these literals always predicate non variable terms (e.g., constants) and are introduced when a term is substituted for a variable whose sortal constraint is not a supersort of that of the term—a matching condition termed *overlapping*.[9] In fact, in Beierle *et al.*'s logic, they also allow overlapping, so when substituting a non variable term t for a variable of sort τ, the SL

[8] Terms are either variables, or of the form $\alpha(\beta_1, \ldots, \beta_n)$ where α is a function symbol and the β_i are terms. If $n = 0$ then the parentheses are usually dispensed with and α is called a constant (symbol).

[9] Most many sorted logics, and certainly all substitutional many sorted logics, insist that the sort of any non variable term substituted for a variable is a subsort of the sort of the variable and that in every well sorted term, the sort of every subterm is a subsort of the sort of the argument position it occupies. If a term has a subterm whose sort is not a subsort of its argument position, then the term may not denote in some interpretations, though it will in others, providing the two sorts have a glb which is not $\bot$. For example, consider substituting a constant c of sort τ_1 for x in the formula $P(f(x))$, where f has been declared to only be defined on arguments of sort τ_2, where the sort structure is as in Figure 1. In any interpretation where c denotes an individual which is in τ_1 and τ_2,

component of the substitution will contain $\tau(t)$. However, when substituting a variable for a variable, there will be characteristic literals in the inferred clause which predicate variables which will not happen in LLAMA.

A further computational problem arises in their logic because, in order to ensure completeness, no wellsortedness checking is done at run time: a valid proof may contain illsorted formulae! Again, this appears to violate the classical methodology of a many sorted logic: classify certain formulae as being nonsense, and thus irrelevant, on syntactic grounds (illsortedness) and reduce the search space by not considering branches containing such formulae. It would appear that the illsorted formulae they wish to consider are those which would be classified as wellsorted by LLAMA through its use of overlapping. However, by not drawing this distinction (and in fact being unable to, since there is no way to express that two sorts are semantically disjoint by using subsort declarations alone) the curtailment of the search space one would naturally hope for cannot be exploited.

We mentioned earlier that this logic also has a rule for resolving characteristic literals, because sort literals can be semantically contradictory without being of the form $\tau(t)$ and $\neg\tau(t)$ as is the case for conventional literals; obviously without such a rule we would lose completeness. Unlike many sorted logics which insist on complete Boolean lattice sort structures, such as LLAMA, where sort literals of arbitrary polarity (i.e., independent of whether the literals are negated or not) can be resolved together, the only possibility (ignoring any unification required) is to resolve a literal of the form $\neg\tau_1(t)$ with one of the form $\tau_2(t)$; these literals will only clash providing τ_2 is a subsort of τ_1. Although one would like to be able to resolve, say Man(c) against Woman(c), given the sort hierarchy in Figure 4, this is not possible, again because the subsort structure does not actually mean that Man and Woman must be semantically disjoint and there is no way to say this within such a sort hierarchy (though of course it could be expressed using axioms containing sort literals in the main body of the axiomatisation).

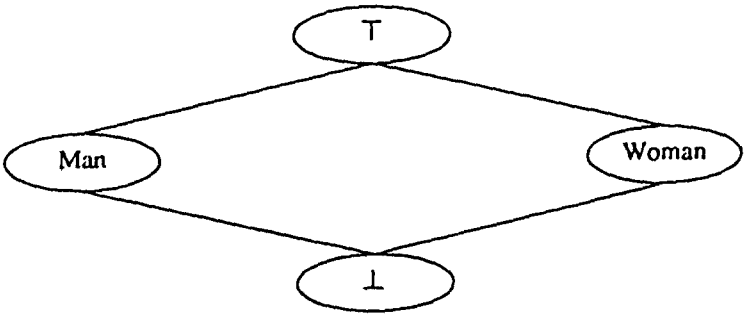

Figure 4. We would like to conclude from this diagram that the sorts Man and Woman are mutually exclusive (referring to disjoint sets). This is not possible in pure subsort hierarchies.

These problems can be avoided if the sort structure is known to be a complete Boolean lattice with $\sqcap$ interpreted as $\cap$, as in LLAMA. The advantages of having such a complete sort structure are several. First, the attachment of sortal constraints to variables as a result of unification can be done in a manner to promote computational efficiency, as discussed above. Secondly, a normal form for clauses can be defined such that no term is predicated by more than one characteristic literal, since any Boolean (logical) combination of sorts is represented by some sort; thus the size of clauses can be reduced. Thirdly, the implementation can take advantage of bit encoding; Aït-Kaci *et al.* [11] have worked out a very clever coding technique to represent large lattices using binary codes, such that lattice theoretic operations can be simply implemented by parallel

then the formula may be true in that interpretation, depending on how f and P are interpreted. But if c denotes a τ_1 which is not a τ_2, then f(c) will fail to denote and $P(f(c))$ must be false. For further discussion see [4]. A better name for this class of formulae may be 'sort consistent.' Thus, *well sorted formulae* are those where every subterm always denotes in every interpretation, *sort consistent* formulae are those where some subterms only denote in some interpretations and *ill sorted formulae* are those where some subterms fail to denote in every interpretation.

bit-wise operations such as 'and' and 'or' directly at the lowest level of the hardware.[10] Finally, reasoning by cases is possible on problems such as Lewis Carrol's 'Salt and Mustard Problem' [12].

However, although desirable for the reasons outlined above, very frequently it will be impossible to specify the sort hierarchy completely in advance. One domain where this has been remarked on is Natural Language understanding [5,13] where new sort information may be acquired during discourse (i.e., the topic may concern the relationship of concepts one would like to think of as sorts). Also, in developing a Naive Physical theory of the commonsense world in the style of Hayes [21,22], Randell and Cohn [23,24] have remarked on the difficulty of extracting a complete sort hierarchy. Although it may be easy to prove (or simply assert) that a particular monadic predicate is non empty, and thus in principle a candidate for being a sort, it is often non trivial to determine the relationship of the potential sort to every other sort (e.g., are they disjoint, is one a subsort of the other?). This problem also arises in the challenge for automatic theorem provers set by Tarver [25]: although he points out three potential monadic predicates as being candidates for sorts in a many sorted theorem proving approach, one of the challenge problems is to prove that one of these implies another: i.e., part of the challenge is to determine the hierarchy!

Thus, we would like to develop a many sorted logic which could benefit from the advantages of a complete Boolean lattice sort structure but at the same time admit of sort structures which are not complete; these incomplete structures may range from structures in which just subsort information is specified, to those in which arbitrary relationships are given between combinations of sorts. However, for simplicity we will first of all consider sort structures in which the only information given is subsort relationships (in the form $\tau_1 \sqsubseteq \tau_2$).

4. EMBEDDING SUBSORT HIERARCHIES IN COMPLETE BOOLEAN LATTICES

The idea we will explore in this section is to embed a partially ordered sort structure specified just by subsort declarations into a complete Boolean lattice, which can then be operated upon by a many sorted logic such as LLAMA which requires (and exploits) a complete Boolean lattice. We will achieve this by inserting sufficient extra sorts below the sorts named in the original hierarchy such that, depending on which of these sorts have empty or non empty interpretations, all possible relationships between the sorts in the original hierarchy can occur. This will be explained further below with the aid of examples. Of course, sorts cannot actually *be* empty, by definition,[11] so, what we must to do is to ensure that no derivation relies on the non emptiness of such sorts without explicitly proving their non emptiness.

Provided the embedding and conditions on non emptiness are correctly specified we should end up with a system that will derive exactly the same set of theorems as a direct inference system such as that of Beierle *et al.* but hopefully in a more computationally efficient manner. Proving this equivalence would also amount to a proof that building a logic which requires complete knowledge about the sort structure is not a practical limit on expressiveness.

Techniques for embedding arbitrary partial orders in true lattices are well known, for example that of Aït-Kaci and Nasr [26]. However, these are minimal embeddings (i.e., the partial order is embedded in the smallest possible lattice) and thus not suitable for our present purpose since the embedding we require must somehow capture all the ambiguity of the simple partial order. An example of the kind of embedding we require will help explain the idea. Suppose we have the hierarchy specified in Figure 1. The smallest complete Boolean lattice in which this can be embedded is illustrated in Figure 5 (given that τ_3 must not be $\bot$ because sorts are nonempty). However, this is not appropriate for our purposes since, if we now interpret greatest lower bounds in the lattice as semantic intersection, since $\tau_1 \sqcap \tau_2 = \tau_3$, we cannot consider the possibility of the denotation of τ_3 being a strict subset of the semantic intersection of τ_1 and τ_2. An embedding which solves this problem is illustrated in Figure 6. It can be seen that in this case τ_3 is a proper subsort of $\tau_1 \sqcap \tau_2$. The reader may now wonder whether we do not now have a new problem

[10] In fact, Aït-Kaci *et al.* give a technique for embedding arbitrary posets into complete Boolean lattices but the embedding is minimal and thus not appropriate for our purposes.

[11] Sorts must be non empty in order to preserve the semantics of quantification: from $\forall x{:}\tau P(x{:}\tau)$ we expect to be able to infer $\exists x{:}\tau P(x{:}\tau)$, which would not be the case if τ could be empty.

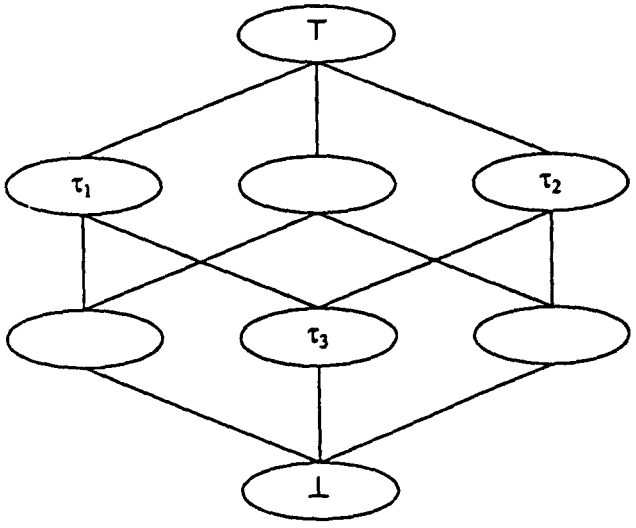

Figure 5. A first attempt to embed the sort poset of Figure 1 into a Boolean lattice, here a 3-cube. This does not allow τ_3 to be distinct from $\tau_1 \sqcap \tau_2$.

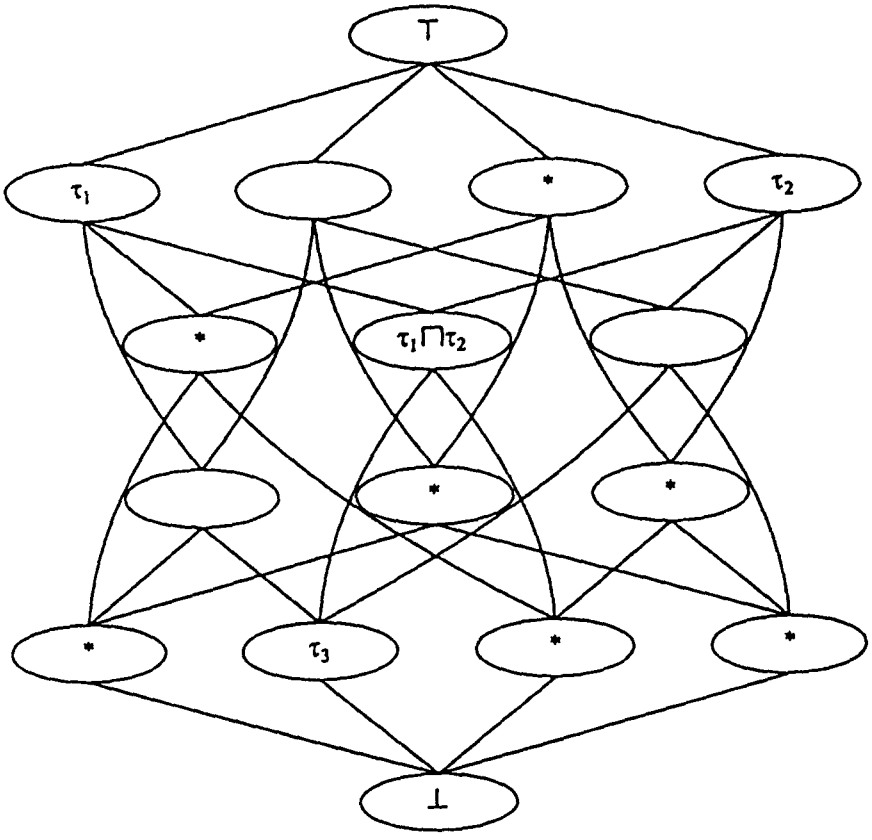

Figure 6. Revised embedding of the sort poset of Figure 1 into a complete Boolean lattice. Here τ_3 is *under* the glb of τ_1 and τ_2. Nodes marked with an asterisk are *potentially empty* sorts. This lattice is a four dimensional hypercube with 2^4 nodes.

on our hands: in any model τ_3 is *forced* to be a strict subset of the semantic intersection of τ_1 and τ_2; in the original subsort hierarchy of Figure 6, either interpretation was possible. However, as already mentioned, we may effectively allow certain of the sorts (those marked with a '*' in Figure 6) to be empty. Consider the base sorts, just above $\bot$, so marked: from left to right, if only the first is empty then $\tau_1 \sqsubseteq \tau_2$, if only the second is empty then $\tau_1 \sqcap \tau_2 = \tau_3$ and if only the third is empty then $\tau_2 \sqsubseteq \tau_1$.

However, in fact the lattice of Figure 6 is still not quite right because it implies that everything in the world is either a τ_1 or a τ_2 (because every base sort is a subsort of τ_1 or τ_2, which is certainly not implied in the subsort hierarchy of Figure 1. In order to solve this problem we must add a further base sort just above $\perp$ to Figure 6 (representing all those individuals which are neither τ_1 nor τ_2) resulting in a complete Boolean lattice with 2^5 nodes. Depending on whether this sort is regarded as empty or not we obtain the two cases of τ_1 and τ_2 covering the universe and this not being the case. We will not attempt to draw this lattice here, but the construction is simple: the extra base sort is disjoint from all nodes in Figure 6 and a further $2^4 - 1$ nodes are added, which are the pairwise least upper bounds of the new base sort with all the nodes except $\perp$ in Figure 6.

Thus in general, given an arbitrary subsort hierarchy S whose sort symbols are the set Δ, then to embed S in a complete Boolean lattice, L, in the manner in which we intend, we need to consider the following set, $\Gamma_1 = \{\{\alpha_1, \ldots, \alpha_n\} \mid \alpha_i \in \{\tau_i, \overline{\tau_i}\}$, where $\Delta = \{\tau_1, \ldots, \tau_n\}$ & $1 \leq i \leq n\}$. We now compute $\Gamma_2 = \{\alpha \mid \alpha \in \Gamma_1$ & $(\tau_1 \sqsubseteq \tau_2) \in S \Longrightarrow \{\tau_1, \overline{\tau_2}\} \not\subseteq \alpha\}$. Γ_1 essentially represents set of all the possible combinations of sorts or their complements (indicated by the bar notation). Γ_2 only contains those sets which are possible given the subsort declarations: if $\tau_1 \sqsubseteq \tau_2$, then we cannot have τ_1 and not τ_2. The elements of Γ_2 all correspond to base nodes in L (i.e., those just above $\perp$). We now identify every element τ of S with the element in L specified by $\sqcup\{\alpha \mid \tau \in \alpha \,\&\, \alpha \in \Gamma_2\}$. This completes the embedding process.

To illustrate this process consider Figure 7 which simply consists of two unrelated sorts τ_1 and τ_2 which we wish to embed in a complete Boolean lattice. The embedding is specified in Figure 8. Γ_1 is $\{\{\tau_1, \tau_2\}\ \{\tau_1, \overline{\tau_2}\},\ \{\overline{\tau_1}, \tau_2\},\ \{\overline{\tau_1}, \overline{\tau_2}\}\}$. Γ_2 is identical to Γ_1 in this case since there is no structure in S. We have labelled the base nodes of the lattice in Figure 8 with the corresponding elements of Γ_2. τ_1 in the original structure is identified with $\sqcup\{\alpha \mid \tau_1 \in \alpha \,\&\, \alpha \in \Gamma_2\}$ $= \{\{\tau_1, \tau_2\}, \{\tau_1, \overline{\tau_2}\}\}$. It can be seen that the node marked as τ_1 in Figure 8 is indeed the lub of the node labelled with $\{\tau_1, \tau_2\}$ and $\{\tau_1, \overline{\tau_2}\}$. The situation for τ_2 is analogous.

As a second example of the embedding process, consider the chain structure sort poset illustrated in Figure 9, where τ_1 is a subsort of τ_2. Γ_1 is the same as before, i.e., $\{\{\tau_1, \tau_2\}\ \{\tau_1, \overline{\tau_2}\},$ $\{\overline{\tau_1}, \tau_2\},\ \{\overline{\tau_1}, \overline{\tau_2}\}\}$, but now Γ_2 is $\{\{\tau_1, \tau_2\}\ \{\tau_1, \overline{\tau_2}\},\ \{\overline{\tau_1}, \overline{\tau_2}\}\}$. The set $\{\overline{\tau_1}, \tau_2\}$ is not in Γ_2 because $\tau_1 \sqsubseteq \tau_2$. Again, we have illustrated the construction by labelling the base nodes in Figure 10 with the elements of Γ_2. τ_1 is identified with $\sqcup\{\{\tau_1, \tau_2\}\} = \{\tau_1, \tau_2\}$ and τ_2 is identified with $\sqcup\{\{\tau_1, \tau_2\}, \{\overline{\tau_1}, \tau_2\}\}$; again, it is easy to verify that the node labelled by τ_2 in Figure 10 is indeed the lub of $\{\tau_1, \tau_2\}$ and $\{\overline{\tau_1}, \tau_2\}$.

5. ENSURING THE CORRECTNESS OF PROOFS

We now turn our attention to ensuring that proofs in the extended logic constitute valid proofs in the original logic. We will make minimal assumptions about the proof system of the extended logic. All that we require is that it allows characteristic literals and assumes a complete Boolean sort lattice. Unifying variables is done by attaching a sort constraint which is the glb of the sorts of the two variables to the unified variable (as we described earlier for standard many sorted logics such as Walther's). Also, we will require a rule[12] for resolving characteristic literals of arbitrary polarity as in LLAMA. We will discuss the rule for evaluating characteristic literals below.[13]

We will distinguish between sorts in the extended lattice which were contained in the original hierarchy (*o_sorts*) and the rest of the sorts (*n_sorts*). If an n_sort is at the base of the lattice (i.e., just above $\perp$) then we also call it a *pe_sort* (potentially empty).[14] Furthermore, any sort

[12]This rule is essentially an instance of *theory resolution* [27].

[13]The logic may be more sophisticated than this; for example, LLAMA has polymorphic descriptions of the non logical symbols, and an evaluation mechanism for arbitrary predicates based on a secondary sort lattice containing four special sorts, UU, TT, FF and EE which have fixed interpretations of true or false, definitely true, definitely false and nonsense, respectively. However, such additional features are not required for the present purpose (unless of course the original logic has such features as well). We will only note here that the mechanism in LLAMA for evaluating characteristic literals requires modification to fit in with the rule described below.

[14]We emphasise again that sorts are not actually empty; what we mean here is that a pe_sort may be empty with respect to the universe of discourse associated with the original signature; by extending the sort hierarchy, we may be forced to add to elements to the universe of discourse in order that these pe_sorts are non empty, but these may be the only elements of the sets these sorts denote.

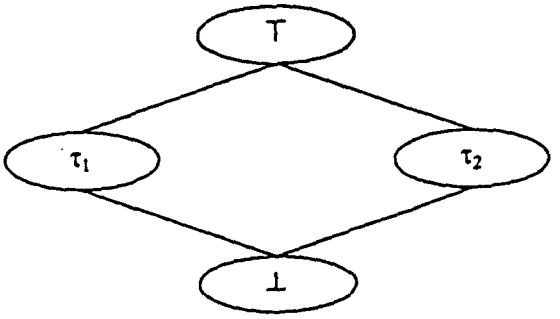

Figure 7. A 'flat' sort poset of 2 independent sorts; there is no structure to reduce the embedding process, so $\Gamma_1 = \Gamma_2$.

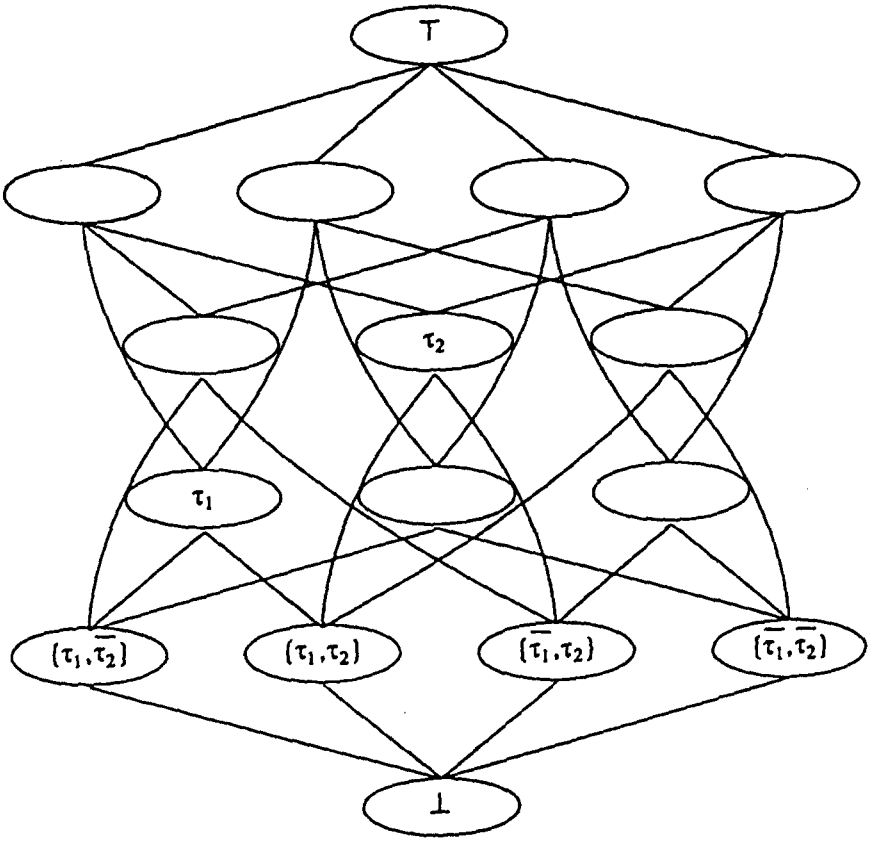

Figure 8. The worst case. A 'flat' sort poset of only two named sorts, τ_1 and τ_2, requires this Boolean lattice of 2^{2^2} nodes. The base nodes are labelled with the corresponding nodes of Γ_2.

which is a least upper bound of pe_sorts is also a pe_sort. Sorts which are not pe_sorts have at least one o_sort beneath them and will call them *ne_sorts* (non empty sorts). In Figure 6 the pe_sorts are marked with a '*'.

Our main concern is with the pe_sorts. We must ensure that no proof mechanism relies on these being nonempty without explicitly proving this to be the case. So when does a proof rely on a sort being non empty? The problem comes when a variable, say $x : \tau_1$ is constrained to be of such a possibly empty sort, i.e., τ_1 is a pe_sort. If a non variable term α is substituted for that variable then there is no problem: by definition, the sort of α, τ_2, cannot be a subsort of the sort of the variable, for the declared sort of any non variable term (e.g., a constant) must be an o_sort by definition, and no subsorts of pe_sorts are o_sorts, so the sorts 'overlap' as described above (i.e., τ_2 is not a subsort of the τ_1 but they have a non $\perp$ glb) and thus a characteristic literal of the form $\neg\tau_1(\alpha)$ will be included in the inferred clause (a *prosthetic literal* in LLAMA's terminology) which effectively predicates the non emptiness of the pe_sort τ_1. This is easy to see in a refutation based calculus where we are attempting to derive the empty clause; clearly

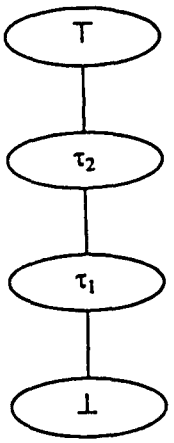

Figure 9. A simple 'chain'-structured sort poset can be embedded in the lattice of Figure 10.

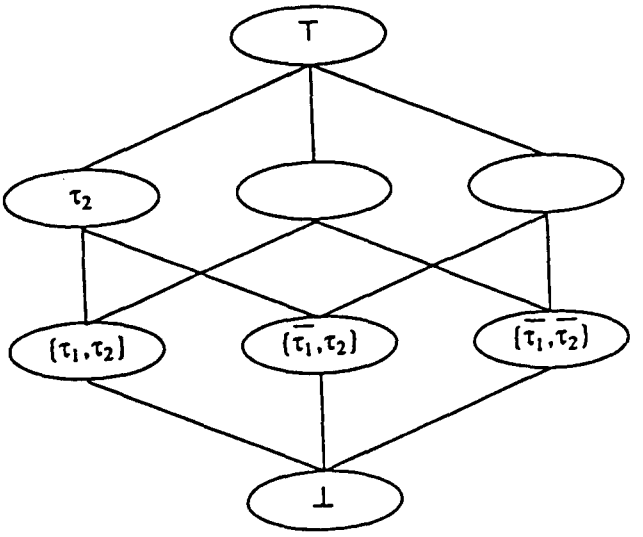

Figure 10. Embedding of Figure 9 into a complete Boolean lattice; a 'chain' of two named sorts requires this lattice of only 2^{2+1} nodes. τ_1 is identified with the base node labelled by $\{\tau_1, \tau_2\}$.

a clause (which is a disjunction of literals) can only be made false by refuting every constituent literal, and the literal $\neg\tau_1(\alpha)$ can only be refuted by proving that α is indeed of sort τ_1 and therefore τ_1 is non empty (because whatever α denotes is in it).

However, suppose we are considering a resolution based calculus, and two clauses are resolved together such that the 'resolved upon literals' contain a variable with a pe_sort constraint, and the variable does not occur in the resolvent, then we have effectively assumed that the associated sort is non empty. For example, suppose we are resolving $P(x\colon \tau_1) \vee \Phi$ (where Φ is some further formula) with $\neg P(y\colon \tau_2)$, then the resolvent will be just the formula Φ. If $\tau_1 \sqcap \tau_2$ is a pe_sort, then this inference will be unsound with respect to the original hierarchy. This can easily be seen if we consider the instances of the two parent clauses from which the inference was made: $P(z\colon \tau_1 \sqcap \tau_2) \vee \Phi$ and $\neg P(z\colon \tau_1 \sqcap \tau_2)$—unless there is an object of sort $\tau_1 \sqcap \tau_2$, the first literal in the first clause is vacuously true (since variables are universally quantified) and the inference is thus unsound since truth cannot be refuted! Of course if Φ contains x as well, then z will still occur in the resolvent and we can delay worrying about the possible emptiness of $\tau_1 \sqcap \tau_2$, since z will be labelled with $\tau_1 \sqcap \tau_2$; if this resolvent forms part of the refutation, then the literal(s) in which z occurs will have to be resolved away, and either z will be unified with a non variable term in which case a prosthetic literal will be added as described in the first case above, or it will be unified with another variable which means the case we are currently considering applies. If x does not occur in Φ, then we must ensure that a literal $\neg\tau(z\colon \tau_1 \sqcap \tau_2)$ is added to the resolvent. This is similar to the literals that were added to the logic of Beierle *et al.*, but here there is only one such literal per variable, and such literals predicating variables will only be present when that is the only occurrence of the variable in the clause. Thus the potentially bad computational effects are virtually non existent here since we do not have to worry about choosing an inappropriate instantiation for x when resolving $\neg\tau(x\colon \tau_1 \sqcap \tau_2)$ away, because, since there are no other occurrences of x, any instantiation will do.

The logic presented here will require an evaluation rule for characteristic literals which takes account of pe_sorts. A literal of the form $\tau_1(\alpha)$ is semantically false just when one of the following conditions hold:

(1) If the literal is of negative polarity and α is a non variable term of sort τ_2, $\tau_2 \sqsubseteq \tau_1$, i.e., $\neg\tau_1(\alpha\colon \tau_2)$.

(2) If the literal is of negative polarity and α is a variable of sort τ_2 and $\tau_1 \sqcap \tau_2 \neq \bot$ and α occurs anywhere elsewhere in the formula and all these other occurrences of α are restricted to $\tau_1 \sqcap \tau_2$. E.g., $\neg\tau_1(\alpha\colon \tau_2) \vee P(\alpha\colon \tau_2)$ can be evaluated to $P(\alpha\colon \tau_1 \sqcap \tau_2)$.

(3) If the literal is of negative polarity and α is a variable of sort τ_2 and $\tau_1 \sqcap \tau_2 \neq \bot$ and α does *not* occur elsewhere in the clause and $\tau_1 \sqcap \tau_2$ is a ne_sort. E.g., $\neg\tau_1(\alpha : \tau_2) \vee P(c)$ can be evaluated to $P(c)$ providing $\tau_1 \sqcap \tau_2$ is a ne_sort.

(4) If the literal is of positive polarity and α is a non variable term of sort τ_2 and $\tau_2 \sqsubseteq \top\backslash\tau_1$, where $\top\backslash\tau_1$ denotes the complement of τ in the sort hierarchy with respect to $\top$.

(5) If the literal is of positive polarity and α is a variable of sort τ_2 which occurs elsewhere in the formula and these occurrences are restricted to sort $\tau_3 = (\top\backslash\tau_1) \sqcap \tau_2$ and τ_3 is a ne_sort.

(6) If the literal is of positive polarity and α is a variable of sort τ_2 and $(\top\backslash\tau_1) \sqcap \tau_2 \neq \bot$ and α does not occur elsewhere in the clause and $(\top\backslash\tau_1) \sqcap \tau_2$ is a ne_sort.

The crucial reason why these rules are correct is the assumption of non emptiness of ne_sorts. Condition (1) is trivially correct from what it means for a non variable term to be of a particular sort. Condition (2) is sound because the condition that α is of sort $\tau_1 \sqcap \tau_2$, implies that $\neg\tau_1(\alpha : \tau_2)$ is false, and this condition on α can be imposed on the formula since α occurs elsewhere. Condition (3) is sound because $\tau_1 \sqcap \tau_2$ is a ne_sort implies that there is an object of sort τ_1 which is also a τ_2 so $\neg\tau_1(\alpha : \tau_2)$ is false. Conditions (4,5,6) are simply the duals of the first three since they deal with positive sort literals, and τ_1 is simply replaced by $\top\backslash\tau_1$ which is the logical complement of τ_1 in the sort hierarchy. That these rules form a complete set, follows from the fact that we have considered all combinations of positive and negative literals, and the term being a variable or non variable, and the term occurring elsewhere in the formulae or not. There are only six cases rather than eight because in the case that the term is not a variable we do not need to consider whether the term occurs elsewhere in the formula. In each condition, inspection reveals that the condition is the weakest that will allow the literal to be evaluated to false.

6. EMBEDDING ARBITRARY HIERARCHIES IN COMPLETE BOOLEAN LATTICES

Defining a hierarchy purely by specifying subsorts is not the only way possible: one can also specify glb, lub and complement relationships. Embedding arbitrary hierarchies in a complete Boolean lattice is little different from the procedure already described for a pure subsort hierarchy, but account must be taken of the constraints provided, (in particular the glb, lub and disjointness information) which was not present in a pure subsort description of the hierarchy.

For example, suppose that S is declared by specifications of the form $\tau_1 \sqsubseteq \tau_2$, $\tau_3 = \tau_1 \sqcap \tau_2$ or $\tau_3 = \tau_1 \sqcup \tau_2$, and Δ is the set of sort symbols in S. To build the complete Boolean lattice, we compute Γ_1 as before (in Section 4) but Γ_2 must be more restrictive. Recall that $\Gamma_1 = \{\{\alpha_1, \ldots, \alpha_n\} \mid \alpha_i \in \{\tau_i, \overline{\tau_i}\}$, where $\Delta = \{\tau_1, \ldots, \tau_n\}$ & $1 \leq i \leq n\}$. We now let

$$\begin{aligned}\Gamma_2 = \{\alpha \mid \alpha \in \Gamma_1 \quad & \& \\ & (\tau_1 \sqsubseteq \tau_2) \in \mathbf{S} \Longrightarrow \{\tau_1, \overline{\tau_2}\} \not\subseteq \alpha \quad \& \\ & (\tau_3 = \tau_1 \sqcap \tau_2) \in \mathbf{S} \Longrightarrow (\{\tau_3, \overline{\tau_1}\} \not\subseteq \alpha \quad \& \quad \{\tau_3, \overline{\tau_2}\} \not\subseteq \alpha \quad \& \quad \{\overline{\tau_3}, \tau_1, \tau_2\} \not\subseteq \alpha) \quad \& \\ & (\tau_3 = \tau_1 \sqcup \tau_2) \in \mathbf{S} \Longrightarrow (\{\tau_1, \overline{\tau_3}\} \not\subseteq \alpha \quad \& \quad \{\tau_2, \overline{\tau_3}\} \not\subseteq \alpha \quad \& \quad \{\tau_3, \overline{\tau_1}, \overline{\tau_2}\} \not\subseteq \alpha)\}.\end{aligned}$$

Note that when performing comparisons with $\not\subseteq$, elements of the form $\bot$ or $\top$ are ignored. Given Γ_2, the construction of L is the same as before, namely every node in the lattice L corresponds to the set $\sqcup\{\alpha \mid \tau \in \alpha \,\&\, \alpha \in \Gamma_2\}$, and each base node directly above $\bot$ in L corresponds to a single set from Γ_2.

If on the other hand S is specified using formulae (i.e., as a first order (unsorted) theory), then the specification of Γ_2 is easier. We will assume S is in clausal form and thus contains elements of the form $\{\tau_1(x), \ldots, \tau_n(x), \tau'_1(x), \ldots, \tau'_m(x)\}$, where the $\tau_i \in \Delta$ and $\tau'_i \in \Delta$ where Δ is the set of sort symbols in S. First we will represent the information in S using the bar notation:

$$\mathbf{S}' = \{\alpha \mid (\overline{\tau} \in \alpha) \text{ iff } \tau(x) \in \beta \quad \& \quad (\tau \in \alpha) \text{ iff } \neg\tau(x) \in \beta, \text{ where } \beta \in \mathbf{S}\}.$$

Thus S' is formed by considering each clause in S, and extracting the sort symbols, with a bar if the literal was positive and without if it was negative. Let Γ_1 be the same as before, i.e.,

$\{\{\alpha_1, \ldots, \alpha_n\} \mid \alpha_i \in \{\tau_i, \overline{\tau_i}\}$, where $\Delta = \{\tau_1, \ldots, \tau_n\} \& 1 \leq i \leq n\}$. We can now easily specify Γ_2. $\Gamma_2 = \{\alpha \mid \alpha \in \Gamma_1 \& \beta \in \mathbf{S}' \Longrightarrow \beta \not\subseteq \alpha\}$. If $|\Gamma_1| = 2^{|\Gamma_2|}$, then S already specified a complete Boolean lattice and the embedding is in fact the identity map. (This assumes that S contains no cycles, i.e., one could not prove both $\tau_1(x) \rightarrow \tau_2(x)$ and $\tau_2(x) \rightarrow \tau_1(x)$ from S unless τ_1 and τ_2 are the same syntactic symbol.)

Specifying S just as a set of clauses, each of arbitrary length and with an arbitrary number of positive and negative literals, is totally general. So we are now able to embed an arbitrary sort hierarchy into a complete Boolean lattice of the appropriate form.

Notice that the embedding also solves the problem of under specified lubs. For example consider the sort hierarchy in Figure 11 where τ_1 and τ_2 are known to be disjoint and $\tau_1 \sqsubseteq \tau_3$ and $\tau_2 \sqsubseteq \tau_3$ and τ_1 and τ_2 cover the universe, i.e., we have the following axioms describing S:

$$\tau_1(x) \rightarrow \tau_3(x),$$
$$\tau_2(x) \rightarrow \tau_3(x),$$
$$\tau_1(x) \rightarrow \neg\tau_2(x),$$
$$\tau_1(x) \vee \tau_2(x).$$

This is correctly embedded in the lattice of Figure 12. Notice that $\tau_3 \neq (\tau_1 \sqcup \tau_2)$ in the embedded lattice; rather $(\tau_1 \sqcup \tau_2) \sqsubset \tau_3$. The calculation involved in this embedding is as follows. Γ_1 is $\{\{\tau_1, \tau_2, \tau_3\}, \{\tau_1, \tau_2, \overline{\tau_3}\}, \{\tau_1, \overline{\tau_2}, \tau_3\}, \{\tau_1, \overline{\tau_2}, \overline{\tau_3}\}, \{\overline{\tau_1}, \tau_2, \tau_3\}, \{\overline{\tau_1}, \tau_2, \overline{\tau_3}\}, \{\overline{\tau_1}, \overline{\tau_2}, \tau_3\}, \{\overline{\tau_1}, \overline{\tau_2}, \overline{\tau_3}\}\}$. All but three of the elements of Γ_1 are eliminated when constructing Γ_2. The elements containing $\{\tau_1, \tau_2\}$ as a subset are eliminated because τ_1 and τ_2 are disjoint. The elements containing $\{\overline{\tau_1}, \overline{\tau_2}\}$ are eliminated because τ_1 and τ_2 cover the universe. Finally, all elements which contain $\overline{\tau_3}$ and either τ_1 or τ_2 are eliminated because of the subsort ordering. This yields Γ_2 as $\{\{\tau_1, \overline{\tau_2}, \tau_3\}, \{\overline{\tau_1}, \tau_2, \tau_3\}, \{\overline{\tau_1}, \overline{\tau_2}, \tau_3\}\}$. These elements label the base nodes of the lattice in Figure 12.

7. COMPLEXITY ISSUES

We have not yet discussed the complexity of the embedding and in particular the space complexity. The worst case is when the original lattice is flat i.e., of the form displayed in Figure 13. Since there are no constraints at all on the possible relationships of the sorts to each other, there will have to be 2^n base sorts in the resulting lattice because then $\Gamma_1 = \Gamma_2$. Thus, for example, when $n = 2$, we obtain the lattice depicted in Figure 8. Using a bit encoding, each base sort takes one bit, so any element of this lattice requires 2^n bits to represent it. So the space required to represent an element of the full lattice would appear to be exponential with respect to the number of nodes in the original hierarchy. In practice there is likely to be at least some structure to the initial hierarchy resulting in a much smaller blowup. For example, given the sort structure in Figure 9 where τ_1 is a subsort of τ_2, the resulting lattice is only of size 2^3, rather than 2^4 as is depicted in Figure 10. In fact, if there is no structure to the sort hierarchy (as in Figure 13) then there is no computational benefit to be derived from using a many sorted logic since every sort will match with every other sort and thus no unifications will fail for sortal reasons.

8. FURTHER WORK AND FINAL COMMENTS

We have deliberately concentrated here on a fairly informal presentation of the ideas. A more theoretical account in which the soundness and completeness arguments sketched very informally here is important. It may also be interesting to relate the embeddings specified here to classical embeddings such as Dedekind-MacNeille completions [28] but the embedding here would seem to be much more complicated.

Aït-Kaci *et al.* [11] present a technique called *modulation*, which allows *much* more compact bit representations of certain kinds of hierarchies embedded in lattices (those where the poset is less structured than a complete Boolean lattice). The impact and application of this technique in the lattices presented here should be investigated.

An interesting idea which springs naturally from this work is to allow empty eponymous (i.e., named) sorts, i.e., drop the standard assumption that user declared sorts are definitely non empty. If a user were to mark certain sorts as possibly empty, then these could be treated much

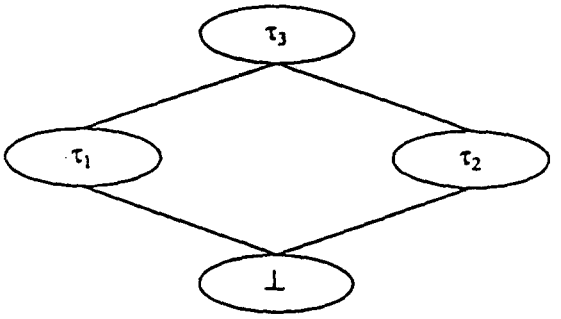

Figure 11. A sort hierarchy with a named least upper bound will be strictly greater than the least upper bound in the embedded lattice in Figure 12.

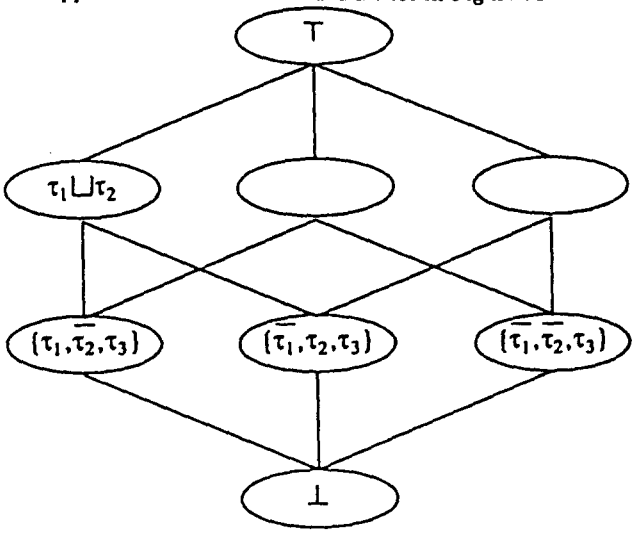

Figure 12. Embedding of Figure 11 into a complete Boolean lattice. τ_1 is identified with the node labelled $\{\tau_1, \overline{\tau_2}, \tau_3\}$, τ_2 is identified with the node labelled $\{\overline{\tau_1}, \tau_2, \tau_3\}$ and τ_3 with $\top$. Notice that $(\tau_1 \sqcup \tau_2) \sqsubset \tau_3$.

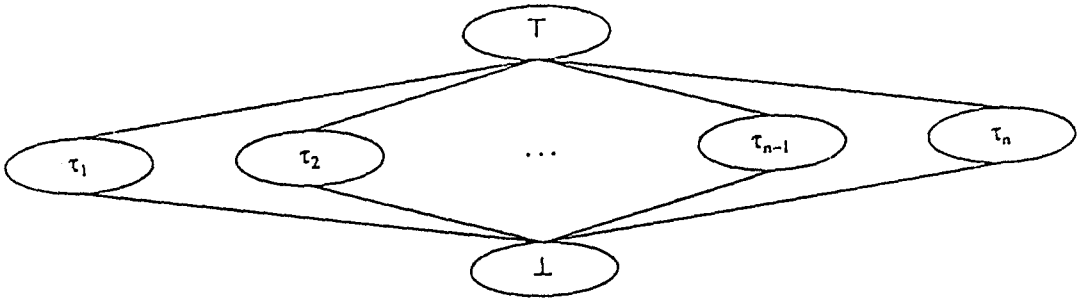

Figure 13. A 'flat' sort poset of n independent sorts; this is the worst case for embedding into a complete Boolean lattice.

as pe_sorts, and be proved to be non empty, on the fly, whenever the proof relies on this. Allowing this possibility would be an important increase in the expressiveness of many sorted logic. Recent work by Weidenbach and Ohlbach [29] has also considered the possibility of a many sorted logic with characteristic literals and possibly empty sorts; their approach is very different and involves abolishing the SBOX, and representing all information in the ABOX. It is not clear whether all the computational advantages associated with many sorted logic are retained by their logic.

We have explored how the structure of a sort hierarchy affects the inference mechanism, and in particular whether an arbitrary hierarchy with under specified glbs, lubs and complements provided by the user can be transformed into a normalized Boolean lattice form for which certain inference mechanisms can be specially tailored. In particular we can obtain an implementation which is in the spirit of a traditional many sorted logic with the minimum use of characteristic literals, but where efficient representations (e.g., bit string) for the hierarchy can be used. This

same embedding method can be used to enhance flexibility and inference capabilities for any reasoning system which relies on poset or lattice structure hierarchies.

REFERENCES

1. J.F. Sowa, *Conceptual Structures*, Addison-Wesley, (1984).
2. R.J. Brachman, R.E. Fikes and H.J. Levesque, Krypton: A functional approach to knowledge representation, *IEEE Computer* 16(10), 67–73 (1983).
3. A.G. Cohn, On the solution of Schubert's steamroller in many sorted logic, *Proc IJCAI 9*, Morgan Kaufmann, Los Altos, 1169–1174 (1985).
4. A.G. Cohn, A more expressive formulation of many sorted logic, *J. Automated Reasoning* 3(2), 113–200 (1987).
5. A. Frisch and J.F. Allen, Knowledge representation and retrieval for natural language processing, TR 104, University of Rochester (1982).
6. H.J. Ohlbach and M. Schmidt-Schauss, The lion and the unicorn, *J. Automated Reasoning* 1(3), 327–332 (1985).
7. C. Walther, A mechanical solution of Schubert's steamroller by many-sorted resolution, *Artificial Intelligence* 26, 217–224 (1985).
8. A.G. Cohn, Taxonomic reasoning with many sorted logics, *Artificial Intelligence Review* 3, 89–128 (1989).
9. H.B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, (1972).
10. J. Herbrand, *Logical Writings*, Harvard Univ. Press, (1971).
11. H. Aït-Kaci, R. Boyer, P. Lincoln and R. Nasr, Efficient implementation of lattice operations, *ACM Trans. on Programming Languages and Systems* 11(1), 115–146 (1989).
12. A.G. Cohn, On the appearance of sortal literals: A non substitutional framework for hybrid reasoning, In *Principles of Representation and Reasoning* (Edited by R.J. Brachman, H.J. Levesque and R. Reiter), Morgan Kaufmann, Los Altos (1989).
13. T. Bollinger, U. Hedstueck and C.R. Rollinger, Reasoning in text understanding: Knowledge processing the LILOG prototype, LILOG Report 49, IBM Germany, Stuttgart (1988).
14. A.M. Frisch, A general framework for sorted deduction: Fundamental results for hybrid reasoning, In *Principles of Representation and Reasoning* (Edited by R. Brachman, H. Levesque & R. Reiter), Morgan Kaufmann, Los Altos (1989).
15. C. Beierle, U. Hedstueck, U. Pletat and J. Siekmann, An order sorted predicate logic with closely coupled taxonomic information, LILOG Report 86, IBM Germany, Stuttgart (1989).
16. R.J. Brachman, B.P. Gilbert and H.J. Levesque, An essential hybrid reasoning system: Knowledge and symbol level accounts of KRYPTON, *Proc. IJCAI*, Morgan Kaufmann, Los Altos, 532–539 (1985).
17. C. Walther, *A Many Sorted Calculus Based on Resolution and Paramodulation*, Pitman, (1987).
18. A.M. Frisch, Parsing with restricted quantification: An initial demonstration, *Computational Intelligence* 2, 142–150 (1986).
19. M. Schmidt-Schauss, A many sorted calculus with polymorphic functions based on resolution and paramodulation, *Proc. IJCAI*, Morgan Kaufmann, Los Altos (1985).
20. A.G. Cohn, Many sorted logic = Unsorted logic + control?, In *Research and Development in Expert Systems III* (Edited by M. Bramer).
21. P.J. Hayes, The second naive physics manifesto, In *Formal Theories of the Common Sense World* (Edited by J.R. Hobbs and R. Moore), pp. 1–36, Ablex (1985).
22. P.J. Hayes, Ontology of liquids, In *Formal Theories of the Commonsense World* (Edited by J. Hobbs and R. Moore), pp. 71–107, Ablex (1985).
23. D. Randell and A.G. Cohn, Modelling topological and metrical properties in physical processes, In *Principles of Representation and Reasoning* (Edited by R.J. Brachman, H.J. Levesque and R. Reiter), Morgan Kaufmann, Los Altos (1989).
24. D.A. Randell and A.G. Cohn, Exploiting temporal and spatial lattice structures, this volume.
25. M. Tarver, *A Challenge Set for Many Sorted Theorem Provers*, University of Leeds, (1990).
26. H. Aït-Kaci and R. Nasr, LOGIN: A logic programming language with built-in inheritance, *J. Logic Programming* 3, 185–215 (1986).
27. M.E. Stickel, Automated deduction by theory resolution, Kluwer, *J. Automated Reasoning* 1, 333–355 (1985).
28. B.A. Davey and H.A. Priestly, *Introduction to Lattices and Order*, Cambridge University Press, (1990).
29. C. Weidenbach and H.J. Ohlbach, A resolution calculus with dynamic sort structures and partial functions, In *Proc. ECAI* (Edited by L.C. Aiello), Pitman, London, pp. 688–693 (1990).